ISSN 1561-5359. Штучний інтелект, 2018, № 4

УДК 004.42

Ю.В. Коваль
Київський національний університет імені Тараса Шевченка, Україна
пр. Глушкова, 4д, м. Київ, 83000

УТОЧНЕННЯ ОСНОВНИХ ПОНЯТЬ ПРОГРАМУВАННЯ ДЛЯ ВІРТУАЛЬНОГО ПРОЦЕСУ

Iu.V. Koval
Taras Shevchenko National University of Kyiv, Ukraine
4d, Hlushkova Ave., Kyiv, 83000

ADJUSTMENT OF BASIC NOTIONS OF PROGRAMMING FOR VIRTUAL PROCESS

У цьому дослідженні зроблено уточнення таких понять як задача, модель, алгоритм, програма, процес та адресований простір, що дозволяє проводити подальші дослідження поняття віртуального процесу. Проведено дослідження деяких з цих понять. Описано операції з елементами пам’яті та процесів і досліджено властивості цих операцій. Наведено поняття функції як елементу програми та процесу, розглянуто способи обміну значеннями між функцією та зовнішнім по відношенню до неї середовищем. Розглянуто властивості цих способів обміну значеннями. Запропоновано класифікацію підпрограм та наведено характеристику функції за цією класифікацією.

Ключові слова: віртуальний процес, програма, алгоритм, модель, задача

Adjustment of problem, model, algorithm, program, process, and address space notions performed in this research that makes possible further research of virtual process notion. Some of this notions are investigated. Operations with memory units and processes described and properties of this operations are investigated. The concept of function as an element of the program or the process is presented, methods of the exchange of values between a function and an external environment with respect to it are proposed. The classification of subroutines is proposed and a description of the function by this classification is given.

Keywords: virtual process, program, algorithm, model, problem

Вступ, постановка проблеми, та мета дослідження

Програмування, як вид діяльності людей, складається з наступних кроків: осмислення задачі, яка має бути вирішена за допомогою програмування, побудова моделі такої задачі, розробка алгоритму, реалізація його у вигляді алгоритмічної програми, компіляція двійкової програми, виконання двійкової програми, отримання результату. Кожен наступний крок має виконуватися за умови відповідності попереднім крокам. Виконання програми може бути доручено комп’ютеру. Комп’ютер, як виконавець програми, здійснює таке виконання за рахунок виконання математичних дій над числовими значеннями, дій з пам’яттю та керуючих дій, що полягають у виборі інших дій, які потрібно виконати. Це обмеження визначає, яку треба будувати модель для задачі, які алгоритми застосовувати та в якій формі буде отримано результат. Формалізація віртуального процесу [1] вимагає уточнення основних понять програмування, дослідження їх властивостей та визначення операцій, які можуть бути до них застосовані, що є проблемою, яка має бути вирішена. Відповідне уточнення вказаних понять для віртуального процесу є метою цього дослідження та буде зроблено в цій статті.

Уточнення поняття задачі

Під задачею в цій статті буде розумітися задача до обчислювального пристрою. Така задача є запитом на отримання відповіді (розв’язання задачі), що може бути отримана шляхом самокерованих обчислень числових значень та дій з пам’яттю.

Задачі для автоматизованих, роботизованих або кіберфізичних систем, що здатні виконувати фізичні дії над об’єктами нашого світу в часі та просторі, заміняються задачами керування складовими цих систем та задачами обліку вико-

наної роботи. Облік виконаної роботи відбувається за допомогою датчиків – пристроїв, що здатні вимірювати деяку фізичну величину та вказувати відповідне числове значення. У такій формі такі задачі зводяться до задач до обчислювального пристрою.

Задачі для не числових значень замінюються задачами з числовими значеннями, де кожному нечисловому значенню ставиться відповідне числове значення. Операції над нечисловими значеннями замінюються діями над відповідними числовими значеннями та пам'яттю.

Далі у тексті цієї статті термін задача буде використовуватися у зазначеному розумінні.

Формально задача має бути описана як множина початкових значень, межі цих значень і значення, що мають бути обчислені та будуть рішенням задачі.

Уточнення поняття моделі

Поняття моделі використовується здавна в науковій та промисловій спільнотах. За рахунок необхідності відповідати вимогам усіх наукових і промислових галузей загальне поняття моделі достатньо розмите [2]. На відміну від загального поняття моделі, моделі, що можуть бути використані в процесі вирішення задач, повинні відповідати щонайменше одній вимозі: інформація про модельовану мову має подаватися в числовій формі або такій, що може бути замінена на числову. Всі дії, які в мовах програмування вищого рівня, ніж мови низького рівня, застосовуються до нечислових значень, і зводяться в мовах низького рівня до обробки числових значень. Пом'якшення цієї жорсткої вимоги на сьогодні відбулося завдяки створенню мов програмування, у яких подання нечислових значень числовими приховано від програміста. Враховуючи наведене, в цій статті під моделлю вважатиметься сукупність характеристик модельованої речі, значення яких можуть бути подані значеннями, що наявні хоча б в одній мові програмування, та залежності між цими значеннями, що можуть бути записані у вигляді виразів чи конструкцій мов програмування. Призначенням моделі є уявне обмірювання модельованої речі з метою вирішення задач стосовно неї.

Розвиток науки, що породжує нові множини значень, нові операції, спричиняє пропорційний розвиток мов програмування, що розширює множину моделей, які можуть бути використані.

У подальшому тексті слово модель буде використовуватися у зазначеному розумінні.

Уточнення поняття алгоритму

Поняття алгоритму [3] взагалі вважається інтуїтивно зрозумілим та не означуваним. Уточнення поняття алгоритм зроблено не один раз [4-7]. Втім, такого уточнення, щоби могло застосовуватись в усіх випадках, поки що немає. У цій статті під алгоритмом буде розумітися спосіб розв'язання задачі, що, беручи до уваги модель, дозволяє знайти розв'язання задачі, виконуючи дії, які можуть бути виконані комп'ютером. Знаходження розв'язання задачі відбувається виконанням алгоритму.

Серед властивостей алгоритму визначимо відтворюваність, що означає можливість багаторазового виконання алгоритму для однієї і тієї самої задачі з одними і тими самими початковими значеннями з отриманням одного і того самого результату, який є розв'язанням задачі. У випадку застосування методів типу Монте-Карло з застосуванням генераторів випадкових чисел, вважатиметься, що розв'язання може бути одне з наперед визначеної множини. Відповідно до теорії ймовірності кожне конкретне розв'язання може бути отримане з певною ймовірністю. Відтворюваність для такого випадку означає, що те саме розв'язання буде отримано, але внаслідок множини спроб виконання алгоритму. Відтворюваність алгоритму не означає, що різні реалізації алгоритму працюватимуть однаково. Реалізація може звужувати множину значень, для якої обчислюється правиль-

не розв'язання. Множина значень, для якої реалізація алгоритму обчислює правильні рішення, є множиною коректної реалізації алгоритму.

Множину значень, що визначають задачу, вказує модель. Під обробкою значень будемо розуміти обчислення допоміжних чи результуючих значень на базі вже обчислених значень чи значень моделі, збереження та зчитування з пам'яті цих значень. Якщо задача допускає декілька рішень, то результатом застосування алгоритму може бути отримання якогось із цих рішень, чи всіх можливих рішень. Інтерактивне обчислення означає, що значення моделі можуть вказуватися під час виконання алгоритму.

Розв'язанням задачі може бути скінченна або нескінченна послідовність значень. У першому випадку від алгоритму очікується скінченна множина дій, що дозволяє обчислити розв'язання задачі. У другому випадку, алгоритм, що розв'язує таку задачу, має працювати нескінченно. Тим не менше, на отримання кожного елемента результуючої послідовності значень алгоритм повинен витрачати скінченну кількість дій. Кількість дій на обчислення одного значення розв'язання не обмежується, що означає потенційну нескінченність дій отримання цього значення.

Як час при виконанні алгоритму можна використовувати кількість виконаних дій. З точки зору практичної придатності, той, хто поставив задачу, сподівається отримати відповідь за прийнятний для нього час. Для узгодження цих двох понять часу використовується час виконання однієї дії. Наслідком цього є те, що для кожної змістовно поставленої задачі існує теоретично розрахована верхня межа кількості дій, за виконання яких необхідно отримати розв'язання задачі. Постійно зростаюча швидкодія комп'ютерів розширює коло задач, що можуть бути розв'язані.

Наступною властивістю є скінченність запису алгоритму. Виділення цієї властивості свідчить про бажання програмістів мати алгоритм як скінченний об'єкт для досліджень. Покрокове виконання алгоритму, що закінчується розв'язанням задачі, породжує послідовність дій, яка розв'язує цю задачу для окремого випадку. Тобто, утворена послідовність дій може розглядатися як алгоритм. Більше того, алгоритми можуть записуватися за допомогою рядків символів, що робить їх придатними для обробки комп'ютером. Отже, запис алгоритму можна змінювати. Такий підхід означає, що алгоритм розв'язання якоїсь задачі як додаткові значення може містити сам себе і змінювати сам себе. Такі самозмінні алгоритми відповідають можливостям комп'ютера змінювати виконуваний код програми та системам з навчанням. Прикладом такої системи є алгоритм поїздки таксі з нерозкритою картою проїздів. Для таксі карта відкривається на невелику відстань – зону прямої видимості. Виконуючи замовлення, таксі збільшує відкриту частину карти, що змінює алгоритм поїздки. Алгоритм припущень дозволяє таксі, не маючи достовірної інформації про проїзд, припустити його існування та переконатися в правдивості припущення. Загальний алгоритм виконання поїздки доповнюється запам'ятованими поїздками, кожна з яких може бути використана як крок алгоритму виконання маршруту. Розумним правилом такого алгоритму є правило знайомої дороги: знайома дорога коротша незнайомої. Слід зауважити, що практика застосування такого алгоритму показує, що давно, або раніше, відкрита карта могла змінитися. Повторне відкриття карти породжує історію зміни карти, що ще більше ускладнює алгоритм.

Таким чином, запис алгоритму є скінченним у кожний момент часу його існування (як під час виконання, так і під час збереження), проте є потенційно нескінченним.

Властивість масовості, яка означає можливість розв'язання задачі для різних початкових значень, є бажаною, проте не

необхідною. Великий клас нескінченних алгоритмів, що забезпечують роботу енергетичних установок чи установок подачі води та їм подібних, завжди виконують одну і ту саму послідовність дій з можливими відхиленнями залежно від зовнішніх обставин.

Уточнення поняття мови програмування та програми

Загальне поняття мови програмування [8] є інтуїтивно зрозумілим. Визначними ознаками мови програмування є такі: мова програмування є штучною мовою, мова програмування призначена для написання програм, мовою програмування можливо записати реалізації моделі задачі та алгоритму її розв'язання.

Мови програмування надають людині можливість керування обчислюючими пристроями, чи пристроями під керуванням обчислюючих пристроїв. У другому випадку все одно відбувається керування обчислюючим пристроєм. Але традиція сприйняття людини та її мозку як єдиного цілого призводить до сприйняття роботизованої чи кіберфізичної системи як єдиного цілого. Серед мов програмування роботизованих чи кіберфізичних систем можливі або вже є звукові, візуальні та тактильні. Всі вони транслюються в якусь внутрішню мову програмування обчислювального пристрою, що є мозком системи.

У цій статті під мовою програмування буде розумітися система позначень, що дозволяє зробити запис, який буде називатися програмою, та засобами якої у цій програмі є можливість реалізовувати моделі та алгоритми. Виконання алгоритму реалізується виконанням програми.

Таким чином, поняття програми для кожної мови програмування визначається незалежно. Тож найбільш загальне поняття програми може бути сформульовано наступними словами: програма – це формальний запис відповідно до вимог однієї або декількох мов програмування, що описує значення, місця їх збереження та способи їх обробки. Програма може бути написана з будь-якою метою. Наприклад, щоби слугувати картиною чи фоновим зображенням, для отримання задоволення, з метою медитації, перевірки певних конструкцій чи прикладів ЯкНеТребаРобити. Синтаксична правильність програми не означає можливості її виконання. Так само вона не означає можливості розв'язання якої-небудь задачі.

На сьогодні достатньо часто зустрічаються програми, що написані декількома мовами програмування. Такий підхід застосовувався і раніше, проте в трохи обмеженій формі. Об'єднання різномовних фрагментів програм відбувалося за рахунок використання бібліотек. Цей підхід застосовується, наприклад, у сімействі компіляторів GCC [9]. Вихідні коди для бібліотечних функцій можуть бути написані різними мовами програмування, проте збиратися в єдину програму для виконання. Це стає можливим за рахунок трансляції в єдину мову програмування низького рівня. Обмеженість такого підходу полягає у необхідності спорідненості мов програмування. Об'єднання в єдину програму кодів мов програмування, що мають різні парадигми, вимагає універсального інтерпретуючого середовища для виконання такої програми. Теоретична можливість створення такого середовища не означає доцільності цього.

Сприйняття низькорівневої програми як єдиного файлу є оманливим та застарілим. Сучасні такі програми можуть використовувати багатофайлову структуру для збереження свого коду.

Визначною особливістю програми для сучасних архітектур комп'ютерів є те, що значення для обробки та дії, що вказують, яка відбуватиметься обробка, зберігаються разом. Принцип інкапсуляції об'єктно орієнтованого програмування прямо використовує та підтримує цю особливість. Єдиний пристрій, що точно відрізняє значення від команди на виконання дій, є виконавець програми. Про-

граміст, що пише програму, весь час дотримується сприйняття коду виконавцем програми і, фактично, так само є її виконавцем. Змінений код програми спочатку є значеннями, а потім стає командами на виконання дії. У подальшому цей код знову може стати значеннями і т.д. Штучне обмеження на заборону зміни коду програми не відміняє цієї можливості. Класичною задачею, що наочно показує цей принцип, є задача написання програми, що генерує сама себе (собі подібну). Використання розв'язання цієї задачі в кодах вірусів показує практичність самої постановки такої задачі.

Уточнення поняття процесу виконання програми

Виконання програми має визначатися дуже точно. При виконанні програми той, хто запустив цю програму на виконання, очікує саме на точне виконання програми. У цій статті під точним виконанням розуміється, по перше, однакове виконання для різних однакових спроб виконання, по друге, точне виконання є точною відповідністю вказівкам для виконання вказаних у програмі. Виконавець програми не має жодної можливості виконати якісь неочікувані дії. Дії, що призводять до помилок виконання так само є очікуваним результатом. Прикладом можуть слугувати: спроба виконати цілочисельне ділення на нуль, спроба прочитати чи записати значення неіснуючого елемента масиву, або спроба відкрити неіснуючий файл. Точність виконання програми для таких випадків визначається не можливістю чи неможливістю виконати ці дії, а наявністю в коді реакції на подібні випадки.

Виконавця програми на сьогодні прийнято називати комп'ютером. Але один і той самий комп'ютер може мати різні операційні системи. Разом на сьогодні це носить назву платформа. Для кожної платформи спосіб виконання програми кожною версією кожної мови програмування визначено однозначно. Конкретну спробу виконання програми на сьогодні прийнято називати процесом. Запуск, або, – іншими словами – створення процесу здійснює, як правило, інший процес. Окрім цього випадку запуск процесу відбувається при запуску операційної системи. По внутрішній структурі процес є сукупністю даних для операційної системи. Виконання процесу відбувається в ті моменти, коли деякі з цих значень, які ми називаємо кодом, сприймаються процесором як команди. Процесор – це та частина комп'ютера, що безпосередньо виконує обчислення. Всі значення, що можуть оброблятися процесором, розташовуються та зберігаються в пристрої, що традиційно називається пам'ять. Отже, пам'ять – це пристрій для зберігання значень. Місце збереження одних значень від інших вирізняється за допомогою адреси. Цей спосіб не є єдиним, проте найбільшою перевагою його є те, що записати значення або визначити, яке значення зберігається в будь-який час для будь-якого елемента пам'яті. Така структура пам'яті відома під назвою пам'ять з прямим доступом. Її головним недоліком є існування максимального розміру пам'яті для кожної архітектури. Іншими видами пам'яті можуть бути послідовна чи стекова пам'ять. Головним їх недоліком є тривалий час на доступ до віддаленого елемента пам'яті. Перевага полягає у можливості мати потенційно нескінченну пам'ять. Такі типи пам'яті використовуються в теоретичних комп'ютерах, наприклад, машині Поста чи машині Тьюрінга.

Має місце сприйняття програмістами такого процесу як програми. Саме тут є джерело сприйняття програми як одиничного об'єкта. Історично програма запам'ятовувалася в пам'яті комп'ютера, а потім виконувалася. Ці дії вже давно автоматизовано, а поняття програми суттєво розширено.

Уточнення поняття адресованого простору та аналіз властивостей такого простору

Адресований простір виникає природнім шляхом як абстракція комп'ютер-

ної пам'яті. Побудова комп'ютерної пам'яті як сукупності однотипних елементів вимагає способу їх ідентифікації. Таким способом, вочевидь, є адресація. Причиною цього є те, що комп'ютер вміє працювати тільки з числами. У адресованому просторі адресація відбувається натуральними числами. Наслідком цього є те, що в цьому просторі легко визначити відношення сусідства. Сусідніми елементами є елементи з сусідніми номерами адрес. Відстань між елементами корелює з кількістю елементів між ними та обчислюється як різниця номерів адрес. Розрізняючи від'ємну та додатну відстань, є можливість визначити поняття напрямку. Природнім результатом розвитку адресації елементів стало утворення масивів (array) та записів (record, structure). У мові програмування С додано накладання (union). Зміна розміру адресного простору визначила перевагу оберненої адресації та зробила її стандартом де-факто [12]. Використання динамічних структур даних призвело до того, що адреса на рівні мов програмування стала звичайним значенням. Не всі мови програмування підтримують відкрите використання адрес, втім, однозначно використовують у реалізаціях похідних технологій, таких як посилання (reference).

Уточнення поняття пам'яті з адресованим простором та аналіз її властивостей

Пам'ять з адресованими елементами обмежена в максимальному розмірі через обмеженість підмножини натуральних чисел, що використовуються в комп'ютерах. Природно, що, чим більший розмір адреси, тим більшу пам'ять можна адресувати за його допомогою.

Означення: відсотковим розміром адреси назвемо відношення у відсотках розміру адреси до розміру адресованого простору.

Теорема 1: Відсотковий розмір адреси в адресованому просторі зменшується при зростанні розміру адреси.

Доведення теореми:

Відсотковий розмір адреси PCAS обчислюється наступною формулою:

PCAS = (AS /US)*100/(2**AS),

де значення AS вказує кількість бітів адреси, що використовується, а значення US вказує кількість бітів одиниці пам'яті. При збільшенні значення AS значення PCAS відповідно до наведеної формули зменшується, що і доводить теорему.

Доведення теореми завершено

Розглянемо приклади для цього значення:

PCAS(8)==0,390625
PCAS(16)==0.0030517578125
PCAS(32)==9.313225746154785e-8
PCAS(64)==4.336808689942018e-17.

Фізична реалізація пам'яті з адресованим простором має на меті створення множини елементів пам'яті з адресацією цих елементів та з підтримкою властивості сусідства. Операції, що можуть застосовуватись до елементів пам'яті, такі: встановлення значення, що зберігається, зчитування значення, яке зберігається; визначення: розміру елемента пам'яті, розміру адреси елемента пам'яті, відстані між елементами пам'яті, кількості проміжних елементів пам'яті. До пам'яті, в цілому, існує операція «визначити розмір пам'яті».

Операції встановлення значення та зчитування значення виконуються гарантовано правильно. Кількість разів виконання цих операцій не обмежується. Операція встановлення значення є ідемпотентною. Операція зчитування значення не руйнує саме значення, що зберігається. Розмір всіх елементів пам'яті має бути однаковим. Розмір адреси елемента пам'яті може відрізнятися від розміру елемента пам'яті. Як правило, розмір адреси більший за розмір елемента пам'яті. У таких випадках адреса зберігається в декількох елементах пам'яті.

Як показує практика, в жодному комп'ютері не використовується фізична пам'ять такого розміру, що відповідає розміру адреси. Окрім цього, використання

декількома процесами спільного адресованого простору породжує небезпеку впливу одного процесу на інший за рахунок зміни значень, що обробляються останнім. Вказані проблеми, а також бажання мати процес, потенційно максимального розміру призвели до застосування технології віртуалізації при реалізації комп'ютерної пам'яті.

Уточнення структури процесу в пам'яті з адресованим простором

Як вже зазначено, з точки зору процесора процес – це набір числових значень. Деякі з цих значень розцінюються як команди. Решта значень поділяється на наступні класи: адресні, константні, глобальні, статичні, автоматичні, динамічні. Адреси та константи – це значення, що розміщені в командах як операнди команд. Враховуючи багаторівневу адресацію, ці значення можуть зберігатися за межами пам'яті, відведеної для команд. Набір елементів пам'яті, що спільно зберігають деяке значення, називається змінною. Операціями до змінної є модифіковані операції до елементів пам'яті та операції створення і знищення. Адреси можуть бути значенням змінних. Змінні поділяються на глобальні, статичні, автоматичні та динамічні. Враховуючи наведене в адресованому просторі процесу, виділяють частину пам'яті для коду (команд) та частину пам'яті для змінних. Друга частина пам'яті поділяється на відповідно глобально/статичну, динамічну та автоматичну. Розмір глобально/статичної пам'яті постійний. Виділення змінній пам'яті в частини динамічної чи автоматичної пам'яті відповідає операції створення змінної. Звільнення такої пам'яті – операції знищення. Відмінність автоматичної та динамічної змінних полягає в джерелі команд на створення та знищення змінних. Для автоматичної пам'яті це є алгоритм виконання програми, а для динамічної – прямі вказівки програміста у вигляді команд у програмі.

Уточнення поняття віртуальної пам'яті з адресованим простором

Під віртуальною пам'яттю з адресованим простором у цій статті розуміється така сукупність елементів пам'яті, що кожен елемент пам'яті вважається фактично існуючим тоді, коли існує фізична пам'ять, що зберігає значення елемента віртуальної пам'яті. Відповідний фізичний елемент називається базовим для віртуального. Розміри елементів віртуальної пам'яті можуть бути більшими, меншими або дорівнювати розміру елементів фізичної пам'яті. Всі операції, що існують для елементів фізичної пам'яті, існують для елементів віртуальної пам'яті. Операція «розмір пам'яті», як правило, не потрібна, оскільки використовується віртуальна пам'ять максимального розміру.

Реалізація операцій для віртуальної пам'яті суттєво відрізняється від реалізації для фізичної пам'яті. Недоліки, що виникають, намагаються компенсувати обмеженнями на час виконання операцій. Так, операції зчитування та встановлення значення стають або гарантованими, або неможливими, залежно від існування базового елемента. Компенсуючим правилом для операції зчитування є правило зчитування нульового значення у випадку відсутності базового елемента. Операцію встановлення значення таким чином компенсувати не вдається. Тому можливі рішення лежать у площині перевірки результату виділення елемента динамічної пам'яті, або в площині генерації виняткової ситуації. Перше рішення, незважаючи на його простоту та зрозумілість, як правило, не застосовується, щоби мати можливість виділяти великі об'єми пам'яті, використовуючи при цьому малу кількість базових елементів. Друге рішення змушує програміста пам'ятати про негарантованість запису. Компромісом можна вважати рішення про обов'язкове виділення базових елементів для коду, глобальної, статичної та автоматичної пам'яті. Динамічна пам'ять залишається не гарантованою. Втім, наявність вільної динамічної пам'яті, за традицією, завжди перевіряється в коді програм.

Теорема 2: Вказана стратегія використання віртуальної пам'яті гарантує ви-

конання операцій зчитування та запису значень та виконання коду програми в цілому.

Доведення теореми:

Запуск процесу не відбудеться, якщо код, глобальні, статичні змінні та автоматична пам'ять не отримають базові елементи пам’яті. Якщо запуск процесу відбувся, то це означає, що базові елементи пам’яті виділено. Заборона зміни коду означає, що всі команди коду доступні на зчитування. Таким чином, код процесу може бути повністю виконано. Виділені базові елементи пам’яті для глобальних та статичних змінних гарантують виконання операцій зчитування та запису для цих змінних. Операції створення та знищення для цих змінних не застосовуються.

Для гарантування роботи автоматичної пам’яті в системах з віртуальною пам’яттю розмір стеку визначається наперед, виділення цієї пам’яті відбувається під час запуску процесу. Неможливість виділити необхідні базові елементи блокує виконання процесу.

Операції з динамічною пам’яттю завжди починаються з виділення пам’яті. Виділення пам’яті відбувається вдало лише за умови виділення базового елемента пам’яті. Таким чином, вдало завершене виділення пам’яті для динамічної змінної означає гарантованість виконання операцій зчитування та запису. Отже, всі можливі випадки розглянуті, що і доводить теорему.

Доведення теореми завершено.

Слід зазначити що фіксований розмір автоматичної пам’яті не є найкращим рішенням, оскільки обмежує глибину рекурсії. Втім, глибина рекурсії завжди буде обмежена, принаймні розміром адресованого простору. У випадку прийнятності аварійного завершення процесу повідомленням про переповнення стеку в довільний момент часу, можлива стратегія, коли автоматична пам’ять є окремим випадком динамічної. Відмінність попередньої стратегії полягає в тому, що переповнення стеку в ній завжди відбуватиметься в один і той самий момент часу виконання програми. Тоді як у модифікованій стратегії це може відбуватися в різні моменти. Модифікована стратегія має перевагу в тих випадках, коли наявний контроль над середовищем, у якому існує процес та є можливість забезпечення максимального розміру доступної фізичної пам’яті.

Механізм зовнішньої віртуальної пам’яті (swap) дозволяє гарантовано виконувати всі операції для віртуальної пам’яті. Недоліком цього підходу є катастрофічне зменшення швидкодії комп’ютера. Незважаючи на це, всі сучасні операційні системи масового вжитку користуються цією стратегією.

Теорема 3: Усі операції зчитування/запису для віртуальної пам’яті з зовнішньою віртуальною пам’яттю гарантовано виконуватимуться за умови, що розмір зовнішньої віртуальної пам’яті дорівнює розміру адресного простору всіх процесів операційної системи.

Доведення теореми виконується розглядом усіх можливих варіантів.

Фактична швидкодія комп’ютера сильно залежить від стратегії, яка визначає, які фрагменти пам’яті копіюються в зовнішню віртуальну пам’ять.

Операція, що перевіряє наявність бази для елемента віртуальної пам’яті, незважаючи на свою надзвичайну корисність, у мовах програмування відсутня. Більше того, мови програмування подібні до Java, взагалі намагаються приховати від програміста механізми керування пам’яттю [10].

Таким чином, процес – це значення, розміщені в адресованій пам’яті комп’ютера під керуванням операційної системи. Код – це такі значення, що вказані процесору як команди для виконання.

Повний час виконання процесу складається з суми чистого часу виконання процесу та часу очікування процесора. Для послідовних систем з багатьма процесами визначено такі способи очікування як багатозадачність (multitasking): зі спів-

працею (cooperative), з витісненням (preemption), з пріоритетами (with priority), реальним часом (real time). Кожен з видів багатозадачності має свої недоліки та переваги. Втім, практика застосування показала, що багатозадачність з витісненням виявилася найбільш прийнятним рішенням. Це рішення застосовується в UNIX-подібних операційних системах. Процес не може керувати тим, як часто йому нададуть час на обчислювання, втім процес може змінити тривалість обчислення. Щоправда тільки в межах, встановлених в операційній системі. Це робиться за допомогою системної функції nice(). Загрозу для таких систем становить програма під назвою форк-бомб. Відповідь на питання про те, чи може ця програма завадити виконанню інших процесів, залежить від співвідношення загальної можливої кількості процесів і часу неперервного виконання одного процесу.

Визначення базових операцій над процесами

Враховуючи наведене раніше, є можливість визначити операції, що можуть бути застосовані до процесів.

Першою такою операцією є створення процесу. Внаслідок виконання цієї операції виділяється частина пам’яті з адресацією та необхідні елементи цієї пам’яті заповнюються потрібними значеннями. Окрім цього, в таблиці операційної системи заносяться необхідні значення про процес.

Друга операція – знищення процесу. При цьому вивільняється пам’ять, що виділялась процесу та змінюються записи операційної системи про цей процес.

Третя операція – завершення роботи процесу. При цьому процес генерує результат роботи процесу для операційної системи. Як правило цей результат називається кодом завершення. Схожість коду завершення та результату обчислення функції фактично визначили те, що в мовах програмування подібних до С програма є функцією main() з додатковими змінними та функціями. Процес, таким чином, фактично, є обчисленням функції main(). Відмінність другої та третьої операцій у тому, що друга – це зовнішня дія над процесом, а третя – це внутрішня дія процесу.

Четверта операція – клонування процесу. Відома в UNIX-подібних операційних системах як fork(). Під час виконання цієї операції виділяється нова пам’ять з новими адресами та створюються нові записи операційної системи про процес.

Треба зауважити, що перша та четверта операції подібні за результатом виконання, проте принципово розбіжні в засобах. У деяких операційних системах надають перевагу першій, у деяких – четвертій.

П'ята операція – заміна коду процесу. Ця операція передбачає зовнішні по відношенню до процесу дії, що призводять до зміни його коду. Дії самого процесу, спрямовані на зміну власного коду, можливі, проте спричиняють загрозу цілісності коду, і тому, на сьогодні, у більшості випадків заборонені.

Шоста операція – зміна активності процесу. Залежно від типу багатозадачності реалізується суттєво різними способами.

Сьома операція – виконання допоміжного процесу. Ця операція вводить ієрархію підпорядкованості процесів. Втім, при виконанні першої чи третьої операцій прийнято визначати підпорядкованість процесів. Тому ця операція реалізується першою чи третьою операціями.

Восьма операція – об’єднання процесів. Мета цієї операції – отримання спільних результатів від декількох процесів. Складність цієї дії зумовлена адресованістю даних. Зрозуміло, що при об’єднанні даних потрібно виконати узгодження їх адресації. Виходячи з технічних складнощів таких дій, ця операція не реалізується. Натомість, використовуються спрощені способи передачі результуючих значень.

Операції над процесами, як правило, не є елементами мов програмування. Їх реалізують функціями, що взаємодіють

з функціями операційної системи.

Способи організації процесів зі спільними даними

Спільні дані для процесів можуть створюватися в різні способи. Серед них такі способи, як спільний адресований простір, спільний фрагмент пам'яті, спільний ресурс.

Процеси зі спільним адресованим простором

Одним із варіантів для реалізації операції об'єднання процесів є процеси зі спільним адресованим простором. У цьому випадку для всіх процесів існує єдиний простір адрес. Такий підхід зменшує об'єми пам'яті, доступні кожному процесу. Іншим недоліком цього підходу є потенційна доступність усіх даних усім процесам. Ця проблема суттєво звужує коло застосування такого рішення. Можливий компроміс полягає у створенні групи процесів операційної системи зі спільним адресним простором, власником та правами безпеки. У сучасних операційних системах таке рішення відоме як нитки виконання (threads) [12]. Компілятор повинен мати повну інформацію про всі адреси і, як наслідок, має розглядати єдину програму як джерело сукупностей таких процесів.

Процеси зі спільним фрагментом пам'яті

У цьому випадку кожен процес може мати власний адресний простір. Цей простір повинен бути віртуальним. Під'єднання спільного фрагменту пам'яті може відбуватися в довільному місці віртуального адресного простору. При компіляції кожній програмі, що буде виконуватись як такий процес, потрібно надати лише однаковий опис значень. Спільні динамічні змінні можуть існувати лише в межах цього фрагменту пам'яті та використовувати відносну адресацію. Такий підхід повністю відповідає вимогам програмістів, проте вимагає суттєвої доробки операційних систем.

Таким чином, процеси зі спільними даними частково здатні бути реалізацією операції об'єднання процесів, проте зі значними обмеженнями.

Процеси зі спільним ресурсом

Окрім значень, що розміщені в адресному просторі, процеси мають можливість оперувати значеннями, що розташовані за межами цих адресних просторів та складають ресурси операційних систем. Наприклад, такими ресурсами можуть бути файлові системи, бази даних, додаткові апаратні пристрої, тощо. Таке оперування зводиться до копіювання чи перенесення значення із зовнішнього місця розташування в деяке місце в межах адресного простору та навпаки. Це надає процесам можливість обмінюватися значеннями при спільному використанні ресурсів. Окремим випадком такої взаємодії є мережева взаємодія, оскільки для забезпечення виконання обміну значеннями використовуються ресурси декількох операційних систем чи зовнішніх процесів.

Функція як автономна складова процесу

Поняття функції в мовах програмування [13] відмінне від такого самого математичного поняття [14]. У програмуванні функція є цілісною частиною коду, що обчислює результуюче значення у відповідь на виклик функції, спираючись на надані та додаткові значення. Чиста функція спирається лише на надані значення. Але величезна кількість функцій використовує додаткові значення. Особливість цих значень полягає в тому, що вони мають можливість зберігатися за межами часу виконання функції.

Визначення способів отримання та передача значення при виконанні функції

Обмін інформацією між функцією та зовнішнім до неї середовищем можливий в такі способи:

1) отримання/передача значення як параметр значення;
2) отримання/передача значення як параметр посилання;
3) отримання/передача значення як параметр функції;
4) передача результату обчислень;
5) отримання/передача значення за допо-

могою зовнішньої змінної;
6) отримання/передача значення за допомогою статичної змінної;
7) отримання/передача значення за допомогою системної функції;
8) передача значення за допомогою генерації сигналу;
9) отримання значення за допомогою обробки сигналу;
10) передача значення за допомогою надсилання повідомлення;
11) отримання значення за допомогою отримування повідомлення;
12) передача значення за допомогою генерації виняткової ситуації;
13) отримання значення за допомогою перехоплення виняткової ситуації;
14) отримання/передача значення поведінкою виконання.

Уточнимо, можливо неочевидні, властивості цих способів передачі значень.

При першому способі отримання значення очевидне. Передача значення можлива тільки у випадку використання адреси як значення параметру.

При другому способі в уяві програміста аргументом є змінна. Розуміючи, що змінна – це частина пам'яті, стає очевидно, що фактичним аргументом є адреса змінної.

При третьому способі функція, яка є параметром, може мати власний спосіб отримання/передачі значення (з/на)зовні.

У четвертому способі відсутність результуючого значення не означає що немає значення, яке можливо повернути як результат. Незважаючи на це, в більшості випадків відсутність результату означає ігнорування будь-якого результуючого значення. Інша особливість цього методу: довгий час вважалося, що обчислення результату відбувається разом із завершенням виконання функції. Однак у мові програмування Пайтон запропоновано підхід, коли функція має автономне середовище виконання, і в такий спосіб може обчислити одиничний частковий результат, призупинивши, а не завершивши, своє виконання [11].

Шостий спосіб використовується для обміну інформацією між різними викликами однієї функції. Треба зауважити, що в цьому випадку сама функція створює для себе зовнішнє середовище. Це зовнішнє середовище є або в часі між послідовними викликами функції, або в адресованому просторі при вкладених рекурсивних викликах.

Специфічним випадком сьомого способу є системні функції виклику допоміжного процесу чи завершення поточного процесу, за допомогою яких функція обмінюється значеннями з контрагентами за межами власного адресованого простору.

Восьмий, дев'ятий, десятий, та одинадцятий способи, як правило, реалізуються з використанням сьомого способу. Втім, їх надзвичайна важливість змушує винести їх в окремі пункти.

Дванадцятий та тринадцятий способи реалізуються засобами мови програмування та притаманні об'єктно орієнтованому програмуванню.

Чотирнадцятий спосіб є самим неочевидним. Втім, зовнішній по відношенню до функції спостерігач, роздивляючись активність функції може отримувати інформацію подібно до азбуки морзе – активність/пауза; довга/коротка пауза; довга/коротка активність; усякі інші специфічні поведінки, або будь-яка інша система кодів. Для отримування інформації в такий спосіб має бути інструмент спостереження активності, який зазвичай реалізується як системна функція.

Класифікація підпрограм

Критерієм для класифікації підпрограм буде наявність або відсутність комбінації властивостей: обчисленого результату, сторонньої дії, рекурсивного обчислення, пам'яті обчислень; призупинки обчислень. Відповідно до першої властивості виділяють функції та процедури. Наявність сторонньої дії не є обов'язковою для процедури. Процедура може передавати значення, використовуючи чотирнадцятий спосіб передачі/отримання значення. Рекурсивність функції чи процедури забез-

печується стековим механізмом обчислювача. Втім, рекурсивні функції/процедури можуть за допомогою статичних змінних моделювати поведінку нерекурсивних. Функції/процедури з пам’яттю можуть реалізовувати цю властивість статичними, глобальними, зовнішніми чи динамічними змінними.

Таким чином, повна назва класифікованої функції, наприклад, може мати такий вигляд: рекурсивна функція зі сторонньою дією та статичною пам’яттю без можливості призупинки обчислень. Зауважимо, що в мові програмування С використовуються саме такі функції.

Методи взаємодії процесів

Можливості взаємодії процесів визначаються операційною системою. Взаємодії процесів слід одразу розділити на два великих класи: взаємодія без обміну значеннями та взаємодія з обміном значеннями. До методів взаємодії без обміну значеннями в цій статті відносяться послідовне та паралельне виконання, запуск та зупинка процесів. У цьому випадку взаємодія зводиться до конкурентного використання спільних ресурсів, таких як пам’ять, процесор, файлова система, комунікаційні порти, різноманітне обладнання.

До методів взаємодії з обміном значеннями в цьому дослідженні віднесено конвеєрну обробку значень, паралельну (одночасну) обробку значень, ієрархічну обробку значень, обробку значень обміном повідомленнями. Відмінність конвеєрної обробки значень від одногілкової ієрархічної полягає в тому, що ієрархічна обробка значень передбачає агрегацію результатів обробки гілок, навіть, якщо ця гілка одна. Операція об’єднання процесів може використовуватись для обміну значеннями як для паралельної обробки значень, так і для ієрархічної.

Узагальнення поняття програми

Як зазначено в [1], віртуальний процес – це сукупність кодів і значень, що можуть виконуватися та оброблятися на різних, навіть архітектурно, обчислювальних системах у спільному просторі з іменуванням пам’яті.

Таким чином, віртуальний процес є процесом розпорошеного виконання багатомовної програми. Вводити поняття віртуальної програми немає потреби, оскільки програма і так є віртуальною сутністю. Багатомовність чи багатопарадигменість програми теж можуть застосовуватись до звичайних програм. Просто тепер треба усвідомити, що програма більше не є одним виконуваним файлом. Означення програми тепер має бути модифіковано до наступного: програма – це сукупність формальних записів відповідно до вимог використаних мов програмування, що описує значення, місця їх збереження та способи обробки, що передбачають узгоджене виконання програми. Необхідність виконання програми, знову ж таки, не є обов’язковою частиною означення програми, оскільки програму могли писати з якоюсь іншою метою, наприклад, для отримання естетичного задоволення або доведення теоретичного твердження. Можливість існування частин програми у вигляді апаратних рішень так само має сприйматися як один із можливих способів запису програми. Використання апаратних нейронних мереж не означає, що вони не є програми.

Уточнення поняття складність програми та його аналіз

Комп’ютерна адресація має ту відмінність від адресації в натуральних числах, що в комп’ютерах застосовується обмежена підмножина натуральних чисел. Наслідком цього, як вже зазначено, є те, що такий параметр як відсотковий розмір адреси має тенденцію до зменшення. Інше поняття, яке також залежить від розміру адресного простору, є поняття складності програми. Складність програми можливо вимірювати в різні способи. Іноді під складністю програми мають на увазі складність обчислення (або виконання) програми. Цей підхід безумовно корисний, проте не надає характеристик розміру та структурі самої програми. Тому слід відокремити ці поняття.

Структурна складність програми визначається кількістю складових програми.

Ці складові можуть бути статичними, як, наприклад, функція main() у мові програмування C, чи динамічними, як, наприклад, функції в мові програмування PHP чи інших інтерпретованих мовах програмування. Крім того, структурна складність може оцінювати кількість та складність значень, якими оперує процес виконання програми.

Визначимо розмірну складність програми як кількість дій, що записані у виконуваному файлі для заданої програми. Цей підхід дозволяє не розрізняти інтерпретовані та компільовані програми. Невелика різниця полягає в тому, що для компільованих програм це, як правило, будуть команди процесора, а для інтерпретованих програм це будуть команди відповідної віртуальної машини. На сьогодні важливим обмеженням на компільовані програми треба вважати неможливість додавати, міняти, знищувати команди, що мають бути виконані. Інтерпретовані мови не так жорстко ставляться до цього питання. Втім, незалежно від наявності чи відсутності цієї можливості, кожен процес виконання будь-якої програми завжди визначить кількість дій, що були виконані за цей час. У кожен момент виконання програми сумарний розмір усіх команд не може перевищувати розміри адресного простору. Оскільки виконання кожної програми на сьогодні обмежене в часі, то для кожної програми можливо порахувати такий показник, як час виконання однієї команди, або однієї структурної одиниці програми. Інший показник, що може бути порахований – середня кількість виконань команди на одне виконання програми.

Теорема: При збільшенні розмірної складності програми зменшується середня кількість виконань команди.

Доведення теореми:

Ця теорема подібна до закону великих чисел. Технологія програмування передбачає послідовність Задача – Модель – Алгоритм – Програма – Процес – Відповідь. Кожна сформульована практична задача має на меті отримати відповідь за прийнятний час. Тобто, існує найбільший прийнятний час виконання програми, що вирішує цю задачу. Швидкість виконання процесорних команд є сталою величиною. Середня кількість виконання однієї команди обчислюється за формулою: AUN=MPPT/(t*SiPC), де AUN – середня кількість виконання команди, MPPT – максимальний час виконання програми, t – час виконання однієї команди, SiPC – розмірна складність програми. Із формули випливає, що чим більше значення розмірної складності програми, тим менше значення середньої кількості виконання команд, що і доводить теорему.

Доведення теореми завершено.

Наслідком цієї теореми є те, що при зростанні розмірної складності програм комп’ютерні системи з фіксованою кількістю процесорів не зможуть вчасно розв’язувати задачі.

Для демонстрації важливості цього питання розглянемо зростання розмірної складності для програми firefox [15] на наведеному графіку.

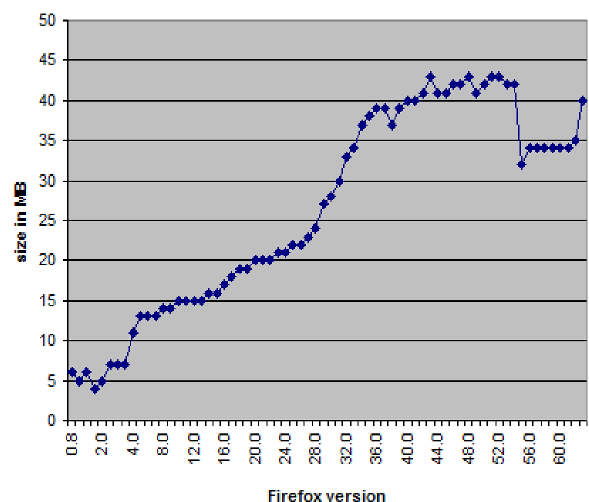


Рис. 1. Графік розміру програми firefox

Висновки

У цій статті зроблено уточнення таких понять як задача, модель, алгоритм, програма, процес та адресований простір з метою проведення подальших досліджень поняття віртуального процесу. Серед усіх можливих моделей обрані ті, що можуть бути побудовані засобами мов програму-

вання. Серед усіх можливих алгоритмів обрано такі, що можуть бути реалізовані засобами мов програмування. Виділено та уточнено такі властивості алгоритмів: відтворюваність, скінченність виконання та запис, масовість. При уточненні поняття «програма» зазначено, що програми всіх рівнів можуть бути багатофайловими або розміщуватися в багатьох місцях та бути заданими з застосуванням декількох мов програмування. Уточнено поняття процесу виконання програми та адресованого простору. Уведено поняття відсоткового адресного розміру та доведено теорему про умови зменшення його значення. Вказано операції для елементів пам'яті та їх властивості. Уточнено структуру процесу в пам'яті з адресованим простором. Уточнено поняття віртуальної пам'яті з адресованим простором. Операції над елементами пам'яті розширено на випадок віртуальної пам'яті та вказано на обмеження, що при цьому утворюються. Наведено стратегію виконання операцій, що забезпечує гарантованість операцій та доведено теорему про це. Розглянуто випадок застосування зовнішньої віртуальної пам'яті, сформульовано та доведено теорему про умову гарантованості операцій над елементами пам'яті в цьому випадку. Визначені операції над процесами. Розглянуто способи організації процесів зі спільними значеннями як спосіб реалізації операції об'єднання процесів. Наведено поняття функції як елемента програми та процесу, розглянуто способи обміну значеннями між функцією та зовнішнім по відношенню до неї середовищем. Розглянуто властивості цих способів обміну значеннями. Запропоновано класифікацію підпрограм та наведено характеристику функції за цією класифікацією. Вказано методи взаємодії процесів. Наведено узагальнення поняття програми та поняття розмірної складності програми. Уведено поняття середньої кількості виконань команди, сформульовано та доведено теорему про умови зменшення її значення.

Література

1. Крак, Ю.В., Коваль, Ю.В., Ставровський, А.Б. (2015) Віртуальний процес: означення та застосування в створенні системи жестового інтерфейсу. *Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки*, 1. С. 141-144.
2. Scientific modelingю *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/scientific-modeling
3. Algorithm. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/algorithm
4. Марков, А.А., Нагорний, Н.М. (1996) Теория алгоритмов. М.:ФАЗИСТ.– 448 с.
5. Мальцев, А.И. (1986) Алгоритм и рекурсивные функции. М.:Наука. – 367 с.
6. Глушков, В.М., Цейтлин, Г.Е., Ющенко, Е.Л. (1985) Алгебра, языки, программирование. К.: Наукова думка. – 327с.
7. Gurevich, Yu. (2012) What is an algorithm? in SOFSEM: Theory and Practice of Computer Science (eds. M. Bielikova et al.), Springer LNCS 7147, pp. 31–42.
8. Computer programming language. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/technology/computer-programming-language
9. GCC, the GNU Compiler Collection [Online]. Available: https://www.gnu.org/software/gcc/
10. Brugali, D., Torchiano, M. (2005) *Software Development: Case Studies in Java* Addisson-Wesley.
11. Rossum, van G. (2006) *Python Reference Manual*.
12. Englander, I. (2009) *The architecture of computer hardware, system software, and networking* Wiley.
13. Function mathematics. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/function-mathematics
14. Function (programming). *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/technology/computer-programming-language/Control-structures
15. Firefox *Mozilla Foundation* [Online]. Available: https://www.mozilla.org/en-US/firefox/

References

1. Krak, Iu.V., Koval, Iu.V., Stavrovskyi, A.B. (2015) Virtualnyi protses: oznachennia ta zastosuvannia v stvorenni systemy zhestovoho interfeisu. *Visnyk Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka Seriia fizyko-matematychni nauky*, 1. S. 141-144.
2. Scientific modelingю *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/scientific-modeling
3. Algorithm. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/algorithm
4. Markov, A.A., Nagorniy, N.M. (1996) Teoriya algoritmov. M.:FAZIST.– 448 s.

5. Maltsev, A.I. (1986) Algoritm i rekursivnyie funktsii. M.:Nauka. – 367 s.
6. Glushkov, V.M., Tseytlin, G.E., Yuschenko, E.L. (1985) Algebra, yazyiki, programmirovanie. K.: Naukova dumka. – 327s.
7. Gurevich, Yu. (2012) What is an algorithm? in SOFSEM: Theory and Practice of Computer Science (eds. M. Bielikova et al.), Springer LNCS 7147, pp. 31–42.
8. Computer programming language. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/technology/computer-programming-language
9. GCC, the GNU Compiler Collection [Online]. Available: https://www.gnu.org/software/gcc/
10. Brugali, D., Torchiano, M. (2005) *Software Development: Case Studies in Java* Addisson-Wesley.
11. Rossum, van G. (2006) *Python Reference Manual*.
12. Englander, I. (2009) *The architecture of computer hardware, system software, and networking Wiley.*
13. Function mathematics. *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/science/function-mathematics
14. Function (programming). *Encyclopedia Britannica* [Online]. Available: https://www.britannica.com/technology/computer-programming-language/Control-structures
15. Firefox *Mozilla Foundation* [Online]. Available: https://www.mozilla.org/en-US/firefox/

RESUME

Iu.V. Koval

Adjustment of basic notions of programming for virtual process

Adjustment of problem, model, algorithm, program, process, and address space notions performed in this research that makes possible further research of virtual process notion. Among all possible models, those that can be built using programming languages are selected. Among all possible algorithms that can be implemented by means of programming languages are selected. The following algorithm properties are specified and adjusted: reproducibility, finiteness of performance and recording, mass. When refining the concept of the program, states that programs of all levels can be multi-file or placed in many places and be given with the use of several programming languages. The concept of the process of the program performing and the address space is adjusted. The concept of percentage address size was proposed and the theorem on the conditions for reducing its value was proved. The structure of the process in memory with the address space is adjusted. The concept of virtual memory with the address space is adjusted. Operations with memory units and processes described and properties of this operations are investigated. The case of using external virtual memory is investigated, the theorem on the condition of the assurance of operations on memory elements in this case is formulated and proved. The methods of organizing processes with common values as a way of realizing the process of combining processes are considered. The concept of function as an element of the program or the process is presented, methods of the exchange of values between a function and an external environment with respect to it are proposed. The classification of subroutines is proposed and a description of the function by this classification is given. The methods of interaction of processes are indicated. The generalization of the concept of the program and the notion of the size complexity of the program is given. The concept of the average number of execution of a command was proposed, and a theorem on the conditions for reducing its value was formulated and proved. As an example, a graph of the growth of complexity for the firefox program is present.

Надійшла до редакції 04.10.2018